Master Thesis

“ Development of verification methods “

Sofia Gkourgkounia

Supervisors : George Stamoulis, Nestoras

Eumorfopoulos, Panagiotis Bozanis

Volos, June 2017

Table of contents

Table of figures

***Abstract*-Advanced microcontroller bus architecture (AMBA) protocol family provides metric-driven verification of protocol compliance, enabling comprehensive testing of interface intellectual property (IP) blocks and System-on-Chip (SoC) designs. This bachelor thesis presents a work aimed to design the AMBA AHB2APB bridge modeled in VHDL hardware description language (HDL) and simulate the results for read and write operation of data and address using the INCISIVE Cadence tool**.

1. Introduction

Embedded system designers have a choice of using a share or point-to-point bus in their designs. Typically, an embedded design will have a general purpose processor, cache, SDRAM, DMA port, and Bridge port to a slower I/O bus, such as the Advanced Micro Controller Bus Architecture (AMBA) Advanced Peripheral Bus (APB). In addition, there might be a port to a DSP processor, or hardware accelerator, common with the increased use of video in many applications. As chip-level device geometries become smaller and smaller, more and more functionality can be added without the concomitant increase in power and cost per die as seen in prior generations.

The Advanced Microcontroller Bus Architecture (AMBA) was introduced by ARM Ltd 1996 and is widely used as the on-chip bus in system on chip (SoC) designs. AMBA is a registered trademark of ARM Ltd. The first AMBA buses were Advanced System Bus (ASB) and Advanced Peripheral Bus (APB). In its 2nd version, AMBA 2, ARM added AMBA High-performance Bus (AHB) that is a single clock-edge protocol. In 2003, ARM introduced the 3rd generation, AMBA 3, including AXI to reach even higher performance interconnects and the Advanced Trace Bus (ATB) as part of the Core Sight on-chip debugs and trace solution. These protocols are today the de-facto standard for 32-bit embedded processors because they are well documented and can be used without royalties. In 2010 the AMBA 4 specifications were introduced starting with AMBA 4 AXI4, then in 2011 extending system wide coherency with AMBA 4 ACE. In 2013 the AMBA 5 CHI (Coherent Hub Interface) specification was introduced, with a re-designed high-speed transport layer and features designed to reduce congestion. The thesis has been organized as follows. The first section contains the description of AXI4-Lite protocol. Second section describes the AXI 4 Stream specification. Third section shows how we used Questasim for synthesis and simulation.

2. AXI protocol

The AMBA AXI protocol supports high-performance, high-frequency system designs. The AXI protocol:

- is suitable for high-bandwidth and low-latency designs
- provides high-frequency operation without using complex bridges
- meets the interface requirements of a wide range of components
- is suitable for memory controllers with high initial access latency
- provides flexibility in the implementation of interconnect architectures
- is backward-compatible with existing AHB and APB interfaces.

The key features of the AXI protocol are:

- separate address/control and data phases
- support for unaligned data transfers, using byte strobes
- uses burst-based transactions with only the start address issued
- separate read and write data channels, that can provide low-cost *Direct Memory Access* (DMA)
- support for issuing multiple outstanding addresses
- support for out-of-order transaction completion
- permits easy addition of register stages to provide timing closure.

The AXI protocol includes the optional extensions that cover signaling for low-power operation. The AXI protocol includes the AXI4-Lite specification, a subset of AXI4 for communication with simpler control register style interfaces within components.

2.1 AXI4-Lite Definition

The AXI-Full specification proposes a different range of important features such as variable data and address bus widths with high bandwidth burst operations. Also, it offers advanced caching support and several transaction assurances and access permissions. While these features offer the user flexibility and control, it is often useful to be provided with a much simpler peripheral which consists of only a subset of these functions. For that reason, a reduced feature variant of the AXI4-Full specification exists in the form of the “AXI4-Lite”.

The AXI4-Lite interconnect provides only necessary interconnect transactions which are required, and high-level capabilities of the interconnect such as burst support, cache support, and variable bit widths for the address and data buses has been removed. The AXI4-Lite interconnect is suitable for applications where simple control and status monitoring capabilities are required for a custom built IP block.

The key functionality of AXI4-Lite operation is:

- all transactions are of burst length 1
- all data accesses use the full width of the data bus

— AXI4-Lite supports a data bus width of 32-bit or 64-bit.

- all accesses are Non-modifiable, Non-bufferable
- Exclusive accesses are not supported.

Both the AXI-Full and the AXI-Lite have five different channels between the Master and the Slave. Data between the master and the slave can move in both directions simultaneously, and data transfer sizes can be different. The AXI4-Full consists of single address with multiple data with a burst transaction up to 256 data beats, but AXI4-Lite provides only 1 data transfer per transaction with 32 bits data width.

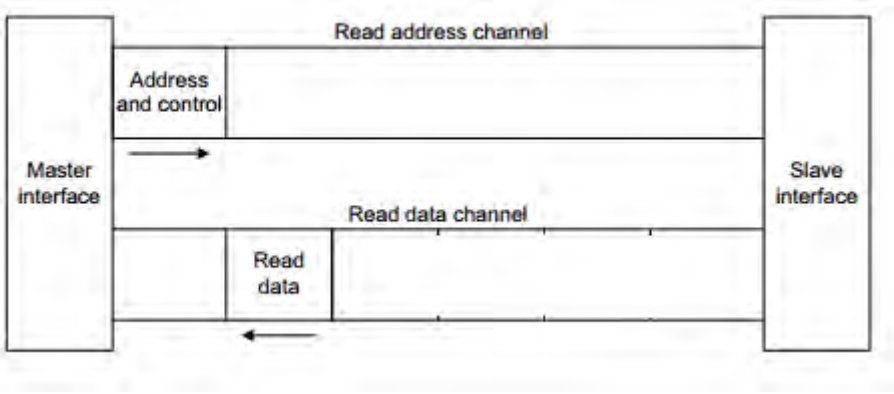

Figure 1: The read architecture of AXI.

A master interface initiates a transaction by specifying a source/target address of the transaction. Simultaneously the master specifies the size of the transaction, information about caching, privileges, QoS, or atomicity properties. There are optional user signals available.

After the transaction is initiated, another phase occurs. If it is a read transaction the slave now starts to send data to the master. In case of a write transaction the master starts to send data to the slave. When the master finishes, the slave returns a response that

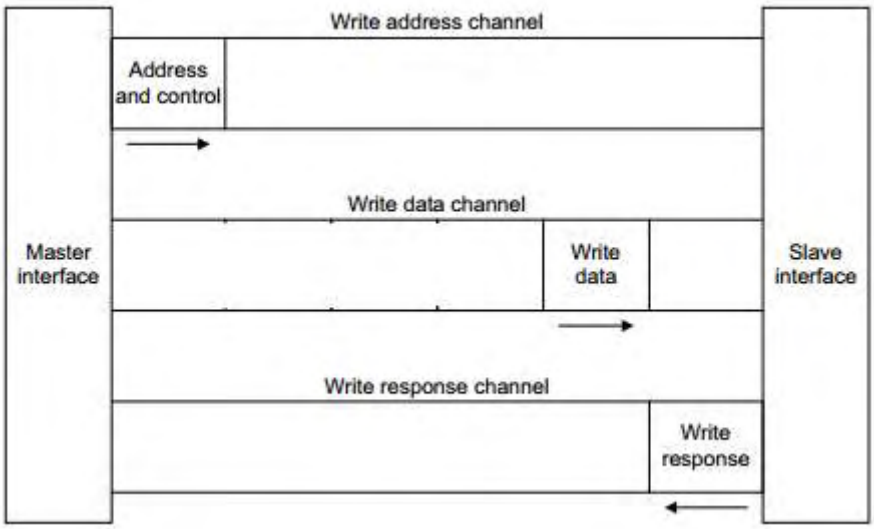

Figure 2: The write architecture of AXI.

allows the master to learn whether the write transaction succeeded or failed. Note that each phase uses a different independent physical channel. Each channel uses handshake signals TVALID and TREADY.

Such a design enables to use pipelining because there is no fixed relationship between the channels. This makes possible to trade-off between cycles of latency and maximum frequency of operation.

2.2 AXI4 Interface: Signaling List and handshaking

Read/write address			*Write data/Write response*			*Read data*		
Name	**Source**	**Width**	**Name**	**Source**	**Width**	**Name**	**Source**	**Width**
AxID	*Master*	-	WID	*Master*	-	RID	*Slave*	-
AxADDR	*Master*	-	WDATA	*Master*	-	RDATA	*Slave*	-
AxLEN	*Master*	8	WSTRB	*Master*	-	RRESP	*Slave*	2
AxSIZE	*Master*	3	WLAST	*Master*	1	RLAST	*Slave*	1
AxBURST	*Master*	2	WUSER	*Master*	-	RUSER	*Slave*	-
AxLOCK	*Master*	2	WVALID	*Master*	1	RVALID	*Slave*	1
AxCACHE	*Master*	4	WREADY	*Slave*	1	RREADY	*Master*	1
AxPROT	*Master*	3						
AxQOS	*Master*	4	BID	*Slave*	-			
AxREGION	*Master*	4	BRESP	*Slave*	2			
AxUSER	*Master*	-	BUSER	*Slave*	-			
AxVALID	*Master*	1	BVALID	*Slave*	1			
AxREADY	*Slave*	1	BREADY	*Master*	1			

Figure 3: AXI4 interface signals

All five transaction channels use the same **VALID/READY** handshake process to transfer address, data, and control information. This two-way flow control mechanism means both the master and slave can control the rate at which the information moves between master and slave. The *source* generates the **VALID** signal to indicate when the address, data or control information is available. The *destination* generates the **READY** signal to indicate that it can accept the information. Transfer occurs only when *both* the **VALID** and **READY** signals are HIGH.

On master and slave interfaces there must be no combinatorial paths between input and output signals.

Each channel has each own **VALID/READY** handshake:

- Address (read/write)
- Data (read/write)
- Response (write only)

It is up to the master to assert the valid signal and the slave to assert the ready signals for all channels except the read data channel where the slave asserts valid to indicate that it is returning data. The agent that asserts ready determines the flexibility as seen in the three waveform options below.

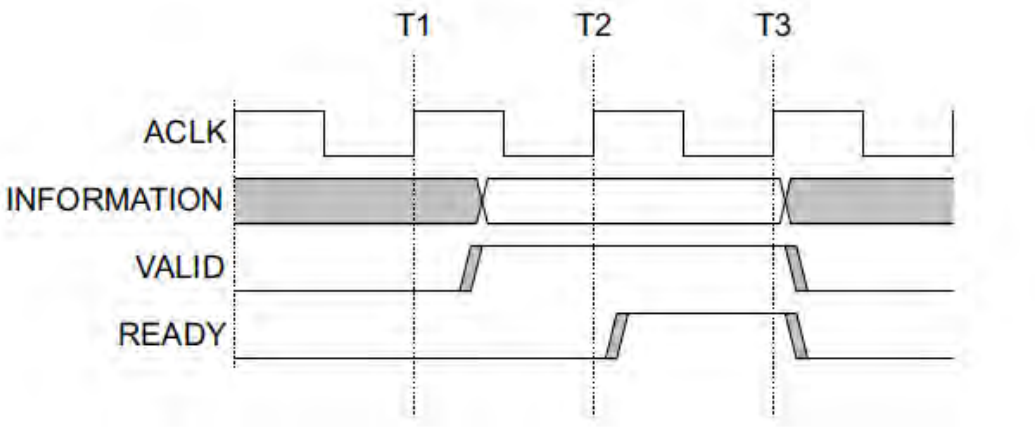

Figure 4: Inserting Wait States (VALID before READY handshake)

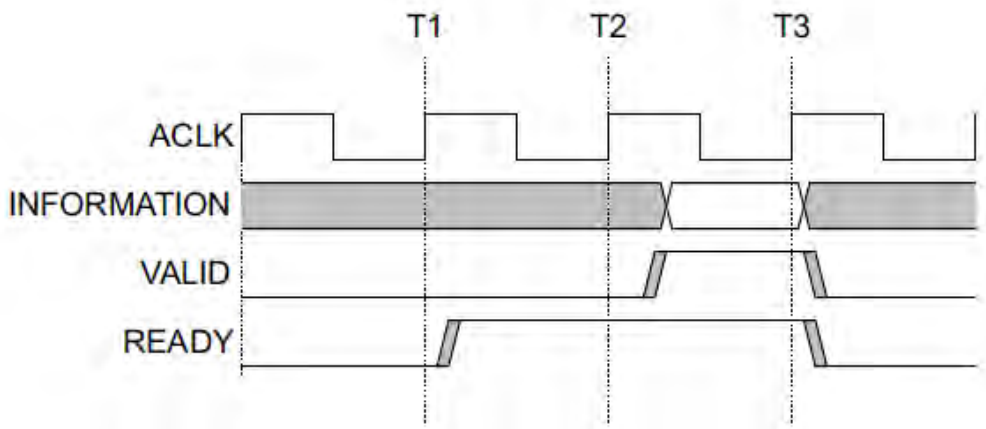

Figure 5: Always Ready (READY before VALID handshake)

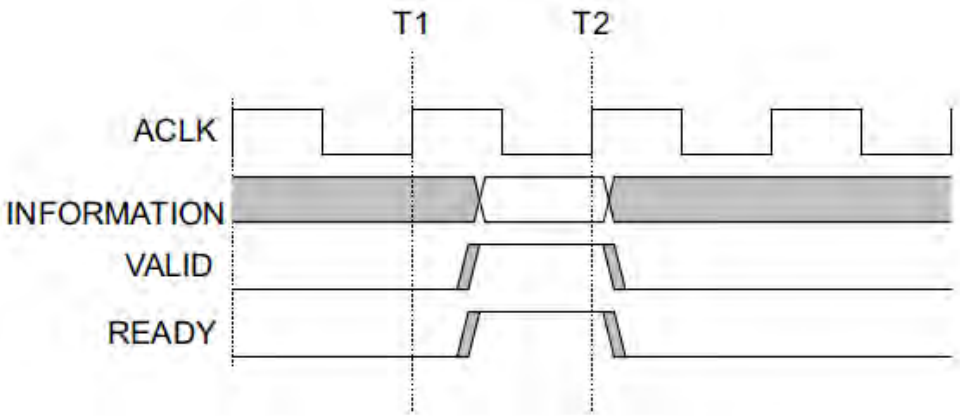

Figure 6: Same Cycle Acknowledge (VALID with READY handshake)

2.3 AXI4 Interface: Read transaction

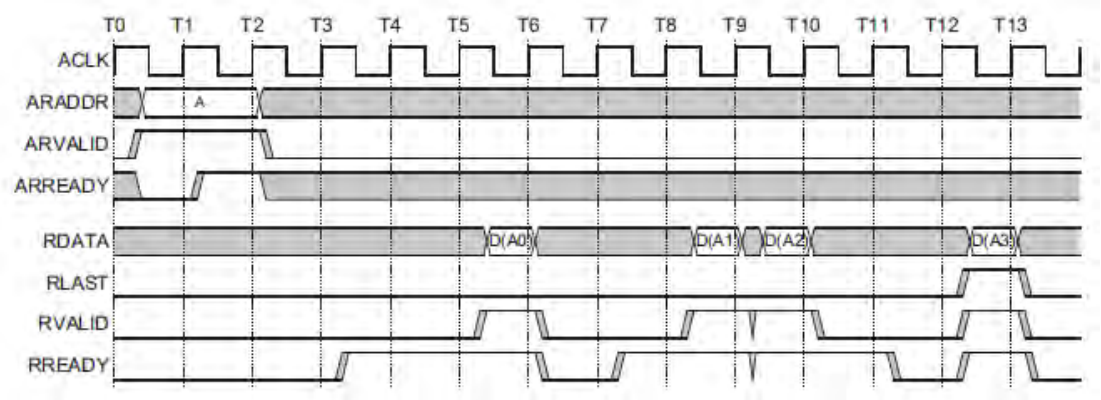

Figure 7: AXI4 - Read Burst

The AXI4 read interfaces allows for data phase transfer up to 256 beats as opposed to the 16 beats that were supported for AXI3. Some signal features may not be required for all types of transfer, depending on user requirements and capabilities.

Read Address Channel:

ARID[3:0] - Master Read address ID. This signal is the identification tag for the read address group of signals.

ARADDR[31:0] - Master Read address. The read address bus gives the initial address of a read burst transaction. Only the start address of the burst is provided and the control signals that are issued alongside the address detail how the address is calculated for the remaining transfers in the burst.

ARLEN[7:0] - Master Burst length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.

ARSIZE[2:0] - Master Burst size. This signal indicates the size of each transfer in the burst.

ARBURST[1:0] - Master Burst type. The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated.

ARLOCK[1:0] - Master Lock type. This signal provides additional information about the atomic characteristics of the transfer.

ARCACHE[3:0] - Master Cache type. This signal provides additional information about the cacheable characteristics of the transfer.

ARPROT[2:0] - Master Protection type. This signal provides protection unit information for the transaction.

ARVALID - Master Read address valid. This signal indicates, when HIGH, that the read address and control information is valid and will remain stable until the address acknowledge signal, ARREADY, is high. 1 = address and control information valid 0 = address and control information not valid.

ARREADY - Slave Read address ready. This signal indicates that the slave is ready to accept an address and associated control signals: 1 = slave ready 0 = slave not ready.

Read Data Channel:

RID[3:0] - Slave Read ID tag. This signal is the ID tag of the read data group of signals. The RID value is generated by the slave and must match the ARID value of the read transaction to which it is responding.

RDATA[31:0] - Slave Read data. The read data bus can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide.

RRESP[1:0] - Slave Read response. This signal indicates the status of the read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.

RLAST - Slave Read last. This signal indicates the last transfer in a read burst.

RVALID - Slave Read valid. This signal indicates that the required read data is available and the read transfer can complete: 1 = read data available 0 = read data not available.

RREADY - Master Read ready. This signal indicates that the master can accept the read data and response information: 1= master ready 0 = master not ready.

2.4 AXI4 Interface: Write transaction

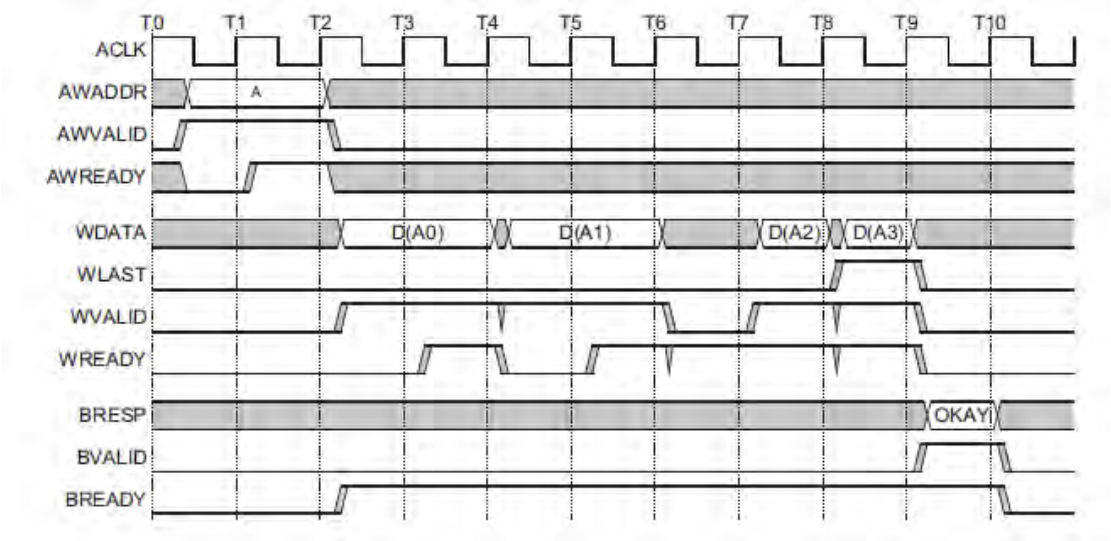

Figure 8: AXI4 - Write Burst

The AXI4 write interfaces allow for data phase transfer up to 256 beats as opposed to the 16 beats that were supported for AXI3. There exist three channels: Address, Data, Response and a selectable data transfer size.

Write Address Channel:

AWID[3:0] - Master Write address ID. This signal is the identification tag for the write address group of signals.

AWADDR[31:0] - Master Write address. The write address bus gives the address of the first transfer in a write burst transaction. The associated control signals are used to determine the addresses of the remaining transfers in the burst.

AWLEN[3:0] - Master Burst length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.

AWSIZE[2:0] - Master Burst size. This signal indicates the size of each transfer in the burst. Byte lane strobes indicate exactly which byte lanes to update.

AWBURST[1:0] - Master Burst type. The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated.

AWLOCK[1:0] - Master Lock type. This signal provides additional information about the atomic characteristics of the transfer.

AWCACHE[3:0] - Master Cache type. This signal indicates the bufferable, cacheable, write-through, write-back, and allocate attributes of the transaction.

AWPROT[2:0] - Master Protection type. This signal indicates the normal, privileged, or secure protection level of the transaction and whether the transaction is a data access or an instruction access.

AWVALID - Master Write address valid. This signal indicates that valid write address and control information are available: 1 = address and control information available 0 = address and control information not available. The address and control information remain stable until the address acknowledge signal, AWREADY, goes HIGH.

AWREADY - Slave Write address ready. This signal indicates that the slave is ready to accept an address and associated control signals: 1 = slave ready 0 = slave not ready.

Write Data Channel:

WID[3:0] - Master Write ID tag. This signal is the ID tag of the write data transfer. The WID value must match the AWID value of the write transaction.

WDATA[31:0] - Master Write data. The write data bus can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide.

WSTRB[3:0] - Master Write strobes. This signal indicates which byte lanes to update in memory. There is one write strobe for each eight bits of the write data bus. Therefore, WSTRB[n] corresponds to WDATA[(8 × n) + 7:(8 × n)].

WLAST - Master Write last. This signal indicates the last transfer in a write burst.

WVALID - Master Write valid. This signal indicates that valid write data and strobes are available: 1 = write data and strobes available 0 = write data and strobes not available.

WREADY - Slave Write ready. This signal indicates that the slave can accept the write data: 1 = slave ready 0 = slave not ready.

Write Response Channel:

BID[3:0] - Slave Response ID. The identification tag of the write response. The BID value must match the AWID value of the write transaction to which the slave is responding.

BRESP[1:0] - Slave Write response. This signal indicates the status of the write transaction. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.

BVALID - Slave Write response valid. This signal indicates that a valid write response is available: 1 = write response available 0 = write response not available.

BREADY - Master Response ready. This signal indicates that the master can accept the response information. 1 = master ready 0 = master not ready.

2.5 AXI4-Lite Signaling List

Global	Write address channel	Write data channel	Write response channel	Read address channel	Read data channel
ACLK	AWVALID	WVALID	BVALID	ARVALID	RVALID
ARESETn	AWREADY	WREADY	BREADY	ARREADY	RREADY
–	AWADDR	WDATA	BRESP	ARADDR	RDATA
–	AWPROT	WSTRB	–	ARPROT	RRESP

Figure 9: AXI4-Lite interface signals

AXI4 signals not supported in AXI4-Lite

The AXI4-Lite interface does not support the following signals:

- **AWLEN, ARLEN:** The burst length is defined to be 1, equivalent to an **AxLEN** value of zero.
- **AWSIZE, ARSIZE:** All accesses are defined to be the width of the data bus.
- **AWBURST, ARBURST:** The burst type has no meaning because the burst length is 1.
- **AWLOCK, ARLOCK:** All accesses are defined as Normal accesses, equivalent to an **AxLOCK** value of zero.
- **AWCACHE, ARCACHE:** All accesses are defined as Non-modifiable, Non-bufferable, equivalent to an **AxCACHE** value of 0b0000.
- **WLAST, RLAST:** All bursts are defined to be of length 1, equivalent to a **WLAST** or **RLAST** value of 1.

2.6 AXI4-Lite Bus Width

AXI4-Lite has a fixed data bus width and all transactions are the same width as the data bus. The data bus width must be, either 32-bits or 64-bits.

ARM expects that:

- the majority of components use a 32-bit interface
- only components requiring 64-bit atomic accesses use a 64-bit interface.

A 64-bit component can be designed for access by 32-bit masters, but the implementation must ensure that the component sees all transactions as 64-bit transactions.

2.7 AXI4-Lite Write Strobes

The AXI4-Lite protocol supports write strobes. This means multi-sized registers can be implemented and also supports memory structures that require support for 8-bit and 16-bit accesses.

All master interfaces and interconnect components must provide correct write strobes.

Any slave component can choose whether to use the write strobes. The options permitted are:

- to make full use of the write strobes
- to ignore the write strobes and treat all write accesses as being the full data bus width
- to detect write strobe combinations that are not supported and provide an error response.

A slave that provides memory access must fully support write strobes. Other slaves in the memory map might support a more limited write strobe option.

When converting from full AXI to AXI4-Lite, a write transaction can be generated on AXI4-Lite with all write strobes deasserted. Automatic suppression of such transactions is permitted but not required.

2.8 AXI4-Lite Optional signaling

AXI4-Lite supports multiple outstanding transactions, but a slave can restrict this by the appropriate use of the handshake signals.

AXI4-Lite does not support AXI IDs. This means all transactions must be in order, and all accesses use a single fixed ID value.

AXI4-Lite does not support data interleaving, the burst length is defined as 1.

3. AXI4-Lite Implementation

The AXI4-Lite IP suite implemented in the scope of this thesis is a solution for the verification of AXI4-lite master and slave devices. The source code of the implementation is System Verilog and the design supports READ and WRITE transaction between N-Masters and N-Slaves.

It supports:

- 1,2, 4,8,16 and 32 bytes data block size
- Multiple outstanding transactions
- Wait states injection
- Full random timings
- Programmable response type
- Read/Write response check
- Full random timings
- Misaligned transfers
- Protection and Cache signals

AXI4-Lite Master commands

1. *ARTransaction(delay, address, protection)* : Read address valid transaction task
2. *RTransaction(delay, data, response)* : Read ready transaction task
3. *AWTransaction(delay, address, protection)* : Write address valid transaction task
4. *WTransaction(delay, data, strobe)* : Write valid transaction task
5. *BTransaction(delay,response)* : Write Response ready transaction task
6. *ReadTransaction(address, protection, data, response)* : Read data transaction task (1,2)
7. *WriteTransaction(address, protection, data, strobe, response)* : Write data transaction task (3,4,5)

AXI4-Lite Slave commands

1. *ARTransaction(delay, address, protection)* : Read address ready transaction task
2. *RTransaction(delay, data, response)* : Read valid transaction task
3. *AWTransaction(delay, address, protection)* : Write address ready transaction task
4. *WTransaction(delay, data, strobe)* : Write ready transaction task
5. *BTransaction(delay,response)* : Write Response valid transaction task
6. *ReadRequest(address, protection)* : Read request transaction task (1)
7. *ReadResponse(data, response)* : Read response transaction task (2)
8. *WriteRequest(address, protection, data, strobe)* : Write request transaction task (3,4)
9. *WriteResponse(response)* : Write response transaction task (5)
10. *RunReadLoop()* : Task to initiate Read transactions. (data initialization, Read Valid response)
11. *RunWriteLoop()* : Task to initiate Write data transactions. (transfer data to strobes)

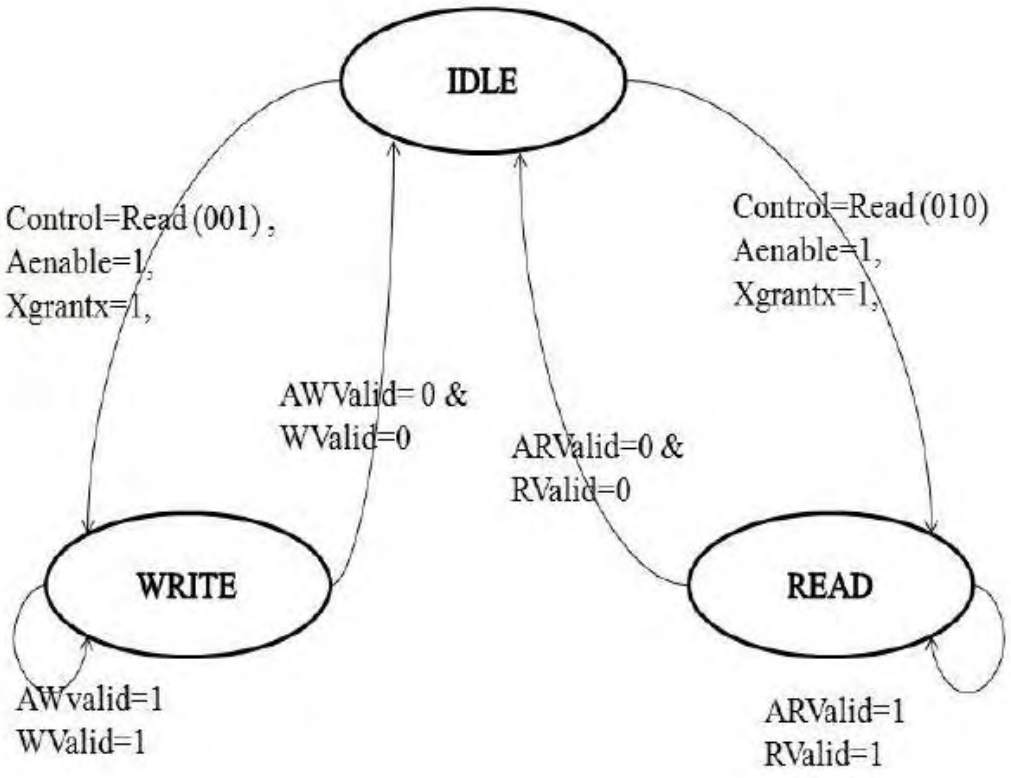

Figure 10: AXI4-Lite IP Suite FSM

4. AXI4-Stream Protocol Specification

The AXI4-Stream protocol is used as a standard interface to connect components that wish to exchange data. The interface can be used to connect a single master, that generates data, to a single slave, that receives data. The protocol can also be used when connecting larger numbers of master and slave components. The protocol supports multiple data streams using the same set of shared wires, allowing a generic interconnect to be constructed that can perform upsizing, downsizing and routing operations.

The AXI4-Stream interface also supports a wide variety of different stream types. The stream protocol defines the association between Transfers and Packets.

The AXI4-Stream protocol is a simplex--one way--bus (a link) from a master to a slave.
There is no way for the slave to respond. It can stop the data ow just by the handshake signals. In fact, a subset of AXI4-Stream is used in the write and read data channels of the AXI4 protocol.

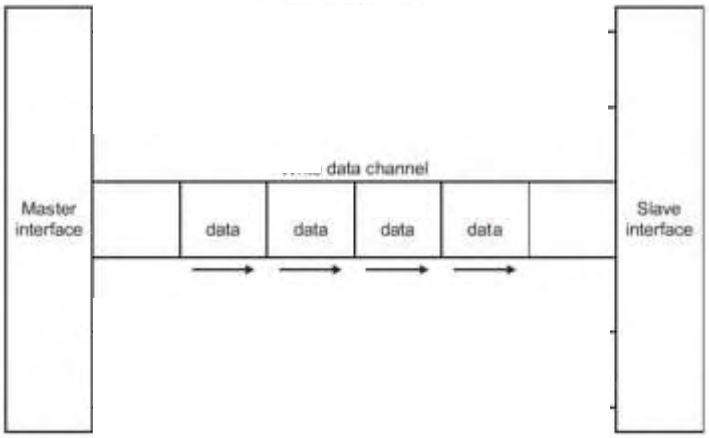

Figure 11: AXI4-Streaming Transfer

AXI4 streaming does not have an address phase; all transactions go to the same place. Note that the direction is always from master to slave. Philosophically this may cloud the concept of master and slave. AXI streaming is very close to the MicroBlaze™ processor FSL except that there is no requirement that a processor be involved.

4.1 AXI4-Stream Interface: Signaling List and handshaking

Name	Source	Width
TVALID	*Master*	1
TREADY	*Slave*	1
TDATA	*Master*	$8n$
TSTRB	*Master*	n
TKEEP	*Master*	n
TLAST	*Master*	1
TID	*Master*	-
TDEST	*Master*	-
TUSER	*Master*	-

Figure 12: AXI4-Stream Interface Signals

All signals except of TVALID and TREADY are optional. There are predefined default values of the signals when any signal is missing. The n represents a number of bytes per data transfer. The signal TKEEP represents a mask of valid bytes in the TDATA signal. The zero bits of the signal marks bytes that can be removed from the stream. It is possible to perform
a transfer where the TKEEP signal contains only zeros (unless there is the TLAST signal asserted). The IP cores are not required to be able to process the zero bytes.
The signal TSTRB is a mask that describes whether the associated byte is a data byte (one) or a position byte (zero). A data byte is a normal valid data byte. A position byte indicates a relative position of data bytes within the stream. The data associated with position bytes is not valid.

The pairs of values of TKEEP and TSTRB have associated semantics:

- TKEEP(i) = 1 ^ TSTRB(i) = 1: the i-th byte is valid and must be transmitted.
- TKEEP(i) = 1 ^ TSTRB(i) = 0: the i-th byte indicates relative position.
- TKEEP(i) = 0 ^ TSTRB(i) = 0: the i-th byte can be removed from the stream.
- TKEEP(i) = 0 ^ TSTRB(i) = 1: represents a forbidden combination.

It is desirable to group bytes into structures called packets for more efficient processing.

A packet is a similar concept to an AXI4 burst. The signal TLAST can be used by the destination to indicate a packet boundary. The protocol does not provide any explicit signaling of the start of a packet.

The signals TID and TDEST provide an identification of a packet transmitted over the stream. This is useful when a unit supports packet interleaving during the transfer. Any processing stage of an AXI-Stream can modify those values. The TID identifies the source of a packet on the link. The signal TDEST provides coarse routing information for the data stream. A routing unit can use the TDEST signal to deliver a packet to the corresponding slave.

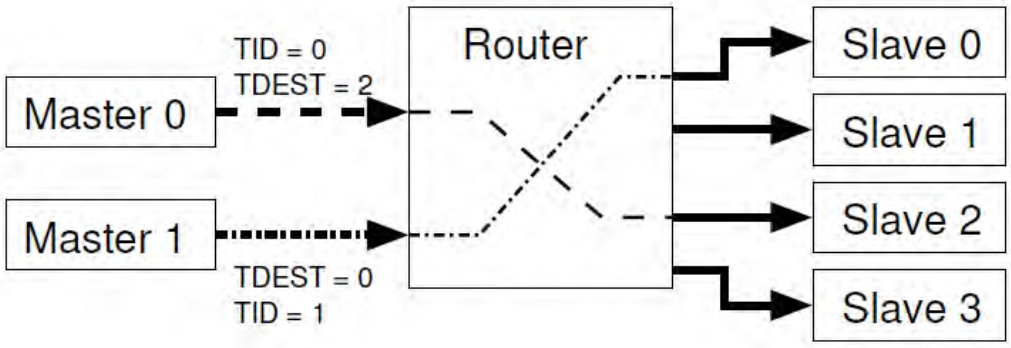

Figure 13: Example of packet routing

Handshake process

The **TVALID** and **TREADY** handshake determines when information is passed across the interface. A two-way flow control mechanism enables both the master and slave to control the
rate at which the data and control information is transmitted across the interface.

For a transfer to occur both the **TVALID** and **TREADY** signals must be asserted. Either **TVALID** or **TREADY** can be asserted first or both can be asserted in the same **ACLK** cycle.

A master is not permitted to wait until **TREADY** is asserted before asserting **TVALID**. Once
TVALID is asserted it must remain asserted until the handshake occurs.

A slave is permitted to wait for **TVALID** to be asserted before asserting the corresponding
TREADY. If a slave asserts **TREADY**, it is permitted to deassert **TREADY** before **TVALID** is asserted.

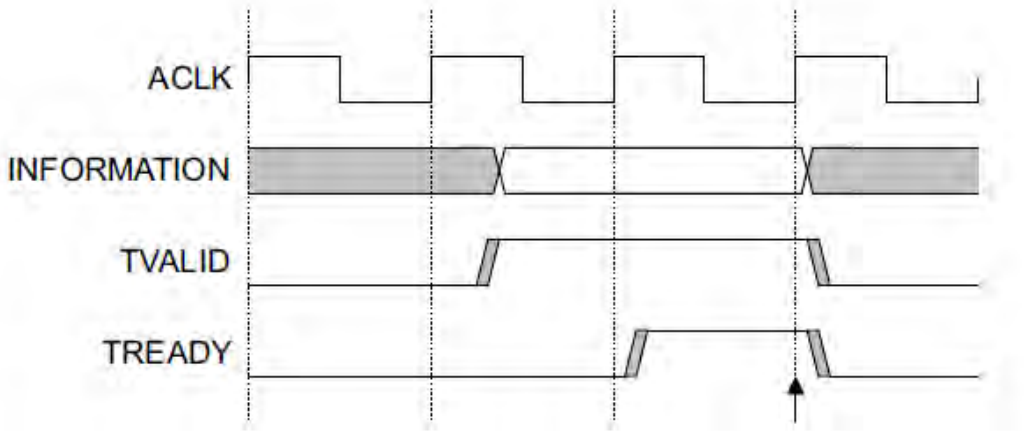

Figure 14: Inserting Wait States (TVALID before TREADY handshake)

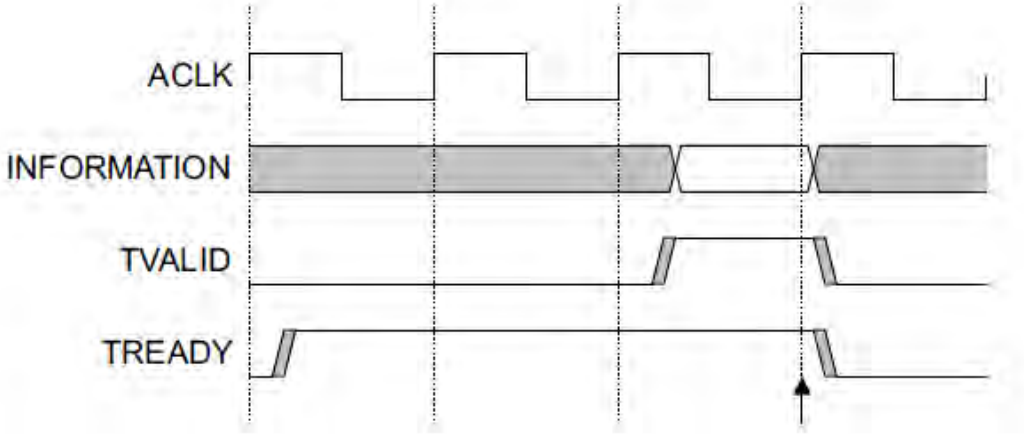

Figure 15: Always Ready (TREADY before TVALID handshake)

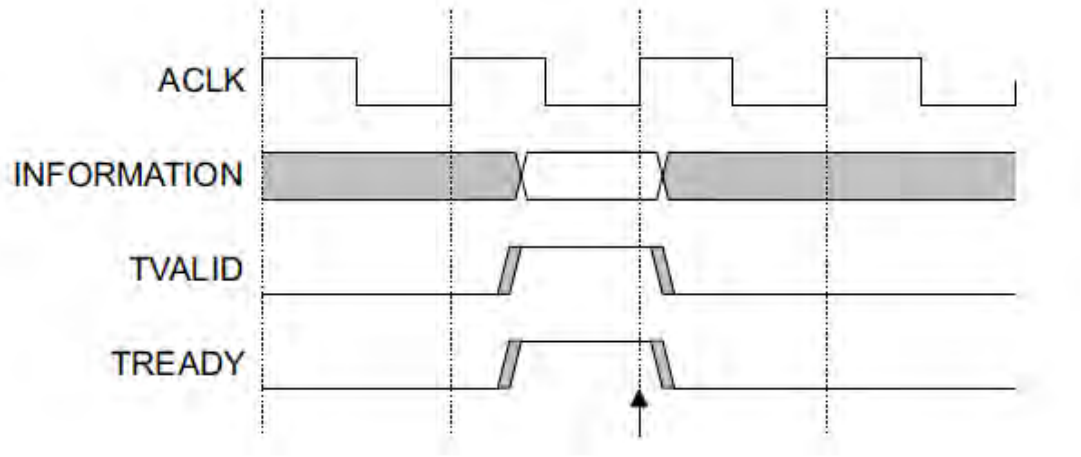

Figure 16: Same Cycle Acknowledge (TVALID with TREADY handshake)

4.2 AXI4-Stream Byte qualifiers

There are two byte qualifiers supported by the AXI4-Stream protocol:

1. **TKEEP** A byte qualifier used to indicate whether the content of the associated byte must be transported to the destination.
2. **TSTRB** A byte qualifier used to indicate whether the content of the associated byte is a data byte or a position byte.

Each bit of **TKEEP** and **TSTRB** is associated with a byte of payload:

- **TKEEP[x]** is associated with **TDATA[(8x+7):8x]**
- **TSTRB[x]** is associated with **TDATA[(8x+7):8x]**

4.3 AXI4-Stream TKEEP qualification

When **TKEEP** is asserted HIGH, it indicates that the associated byte must be transmitted to the destination. When **TKEEP** is deasserted LOW, it indicates a null byte that can be removed from the stream.

It is legal to have a transfer that has all **TKEEP** bits deasserted LOW. It is permissible for a transfer that has all **TKEEP** bits deasserted LOW to be suppressed unless it has **TLAST** asserted HIGH.

It is not mandatory for masters and slaves to handle null bytes, therefore any interconnect that is capable of inserting null bytes in a stream should also be capable of removing them before the stream arrives at a destination that is not capable of handling null bytes.

4.3 AXI4-Stream TSTRB qualification

When **TKEEP** is asserted, **TSTRB** is used to indicate whether the associated byte is a data byte or a position byte. When **TSTRB** is asserted HIGH it indicates that the associated byte contains valid information, and is a data byte.

When **TSTRB** is deasserted LOW it indicates that the associated byte does not contain valid information and is a position byte. A position byte is used to indicate the correct relative position of the data bytes within the stream. Position bytes are typically used when the data stream is performing a partial update of information at the destination. Since the data associated with a position byte is not valid, an interconnect need not transmit the **TDATA** associated with a byte for which **TSTRB** is deasserted LOW.

4.4 AXI4-Stream Packet Boundaries

A packet is a grouping of bytes that are transmitted together across the interface. Infrastructure components can typically be made more efficient by dealing with transfers that are grouped together in packets. An AXI4-Stream packet is similar to an AXI4 burst.

The signals to be considered during a packet transfer are **TID**, **TDEST**, and **TLAST.**

The uses of **TLAST** are:

- when deasserted, **TLAST** indicates that another transfer can follow and therefore it is acceptable to delay the current transfer for the purpose of upsizing, downsizing, or merging
- when asserted, **TLAST** can be used by a destination to indicate a packet boundary
- when asserted, **TLAST** indicates an efficient point to make an arbitration change on a shared link.

TLAST can be used to transmit information between the source and destination. The number of packets, and the number of assertions of **TLAST**, must be preserved between the master and slave.

No explicit signaling of the start of a packet boundary is given in the protocol. The start of a packet is determined as:

- the first occurrence of a **TID** and **TDEST** pair after reset
- the first transfer after the end of the preceding packet for any unique set of **TID** and **TDEST** values.

All bytes within a packet are from the same source and for the same destination and have the same **TID** and **TDEST** values.

The merging of transfers that belong to different packets is not permitted. This requires that two transfers with the same **TID** and **TDEST** values must not be merged if the earlier transfer has **TLAST** asserted.

The merging of transfers with different **TID** or **TDEST** values is never permitted.

4.5 AXI4-Stream Transfer with zero data or position byte

A transfer can have **TLAST** asserted but contain no data or position bytes. This can be used to:

- indicate the end of a packet when there are no more data or position bytes to transmit
- push through any data that is held in intermediate buffers
- complete an operation at an end-point that is expecting a **TLAST** at the end of a packet.

A transfer that has **TLAST** asserted, but does not have any data or position bytes, can be merged with an earlier transfer with matching **TID** and **TDEST** values that does not also have **TLAST** asserted.

Because reordering is not supported, sending a transfer with zero data bytes will effectively push through all transfers between a master and slave.

4.5 AXI4-Stream User Signaling

Typical uses of a streaming interface require some User sideband signaling. Sideband signaling can be used for data byte, transfer, packet, or frame-based information.

There are several uses of User signaling. For example:

- marking the location or type of special data items

• providing ancillary information that must accompany the data, such as parity, control signals, and flags
• identifying segments of a packet.

To ensure a consistent method of transporting User information the protocol defines that User signaling is transferred on a byte basis.

It is recommended, but not required, that the number of **TUSER** bits is an integer multiple of the width of the interface in bytes. The User signals for each byte must be packed together in adjacent bits within **TUSER**.

The location of the User bits is defined as:
• each data byte has m User signals associated with it
• the total width of the interface is n bytes,
• the total number of User bits is u, where u = m * n

The user signals for byte x, where x = 0 ... (n-1), are located at:

TUSER[((x*m)+(m-1)):(x*m)]

The transfer of **TUSER** bits, when the associated **TKEEP** signal is deasserted LOW, is not required or guaranteed.

User bits associated with a null byte, as indicated by the associated **TKEEP** bit, must be removed from the data stream if the null byte is removed from the stream. If a null byte is inserted in the data stream the appropriate number of User bits must also be inserted. When inserting additional bits they must be fixed LOW.

TUSER can be used to convey information that is relevant to an entire transfer rather than to individual bytes. An example of this is where the same information applies to every byte in a transfer and it is more efficient to indicate the additional information once only for the entire transfer rather than replicating it for each byte within the transfer.

TUSER can be used to convey transfer based information but the transport mechanism will divide the **TUSER** information between the data bytes being transported. Reliable transport of transfer-based **TUSER** information can only be guaranteed under the following constraints:

• the data bus width at the input to the interconnect must match the data bus width at the output of the interconnect
• any data width conversion that occurs in the interconnect must not modify the packing of the data between the input to the interconnect and the output of the interconnect

4.5 AXI4-Stream User Signaling

The AXI4-Stream interface has many similarities to an AXI4 write data channel. However,
there are some key differences. These differences are summarized as:

• the AXI4 write data channel does not permit interleaving
• the AXI4-Stream interface does not have a defined or maximum burst or packet length
• the AXI4-Stream interface allows the data width to be any integer number of data bytes
• the AXI4-Stream interface includes **TID** and **TDEST** signals to indicate the source and destination respectively
• the AXI4-Stream interface defines more precisely the manipulation of the **TUSER** sideband signals
• the AXI4-Stream interface includes **TKEEP** signals to allow the insertion and removal of null bytes.

5. AXI4-Stream Implementation

The AXI4-Stream suite implemented in the scope of this thesis is a solution for the verification of AXI4-Stream master and slave devices. The source code of the implementation is System Verilog and operates as a master or a slave, along with the protocol bus monitor.

It supports:

- 1,2, 4,8 and 16 bytes data block size
- Up to 8 user bits per byte
- Wait states injection
- Full random timings
- Programmable response type

Limitations:

- Doesn't support packets with position bytes

AXI4-Stream Master Commands

1. *sendData(inbuff, setLast, tid, tdest)* : Sends data buffer
2. *genSingleTransfer(tdata,tkeep,tstrb,tlast,tuser,tid,tdest)* : Generate single transfer on the AXI4-Stream bus.
3. *createUserBuf(inBuf)* : Create the user defined buffer which will be transmitted alongside the data stream. The information of this buffer will be transmitted via *TUSER* bus when *sendData()* command is used. If the size of this buffer is less than transmitted data buffer the 0s will be transmitted.
4. *busIdle(idleCycles)* : Holds the bus in the idle state for the specified clock cycles
5. *waitCommandDone()* : Wait until all transactions in the buffer are finished
6. *startEnv()* : Start the AXI4-Stream master environment. Don't use data transfer commands before the environment start.

AXI4-Stream Slave Commands

1. *getSingleTransfer(tdata,tkeep,tstrb,tlast,tuser,tid,tdest)* : Get single transfer on the AXI4-Stream bus.
2. *readData(outBuff)* : Reads the complete data packet from the bus.
3. *readUserBuf(outBuff)* : Read the user data.
4. *getTID_TDEST(tid, tdest)* : Returns the values from the *tdest* and *tid* buses.
5. *startEnv()* : Start the AXI4-Stream slave environment

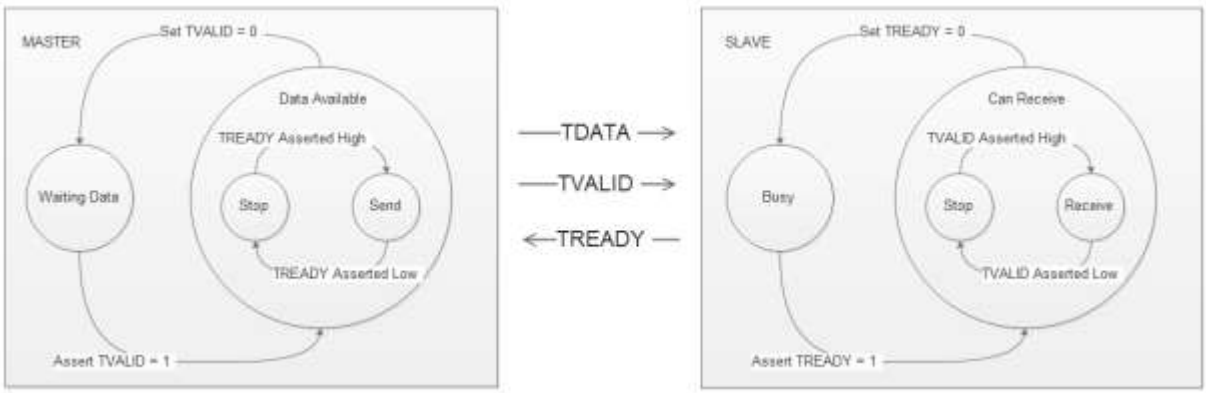

Figure 17: AXI4-Stream State Machine

6. Design integration and Simulation

For both AXI4-Lite and AXI4-Stream IP Suites, Questa Advanced Simulator was used for synthesis and simulation.

AXI4-Lite simulation

For the simulation of the AXI4-Lite design the following test scenario was created:

- 1 AXI4-Lite Master interface
- 1 AXI4-Lite Slave interface
- Insertion of random delays at both Master and Slave
- Generation of random packets
- Master: Writes Data – Gets Write Response – Reads Data – Gets Data
- Slave: Puts Data

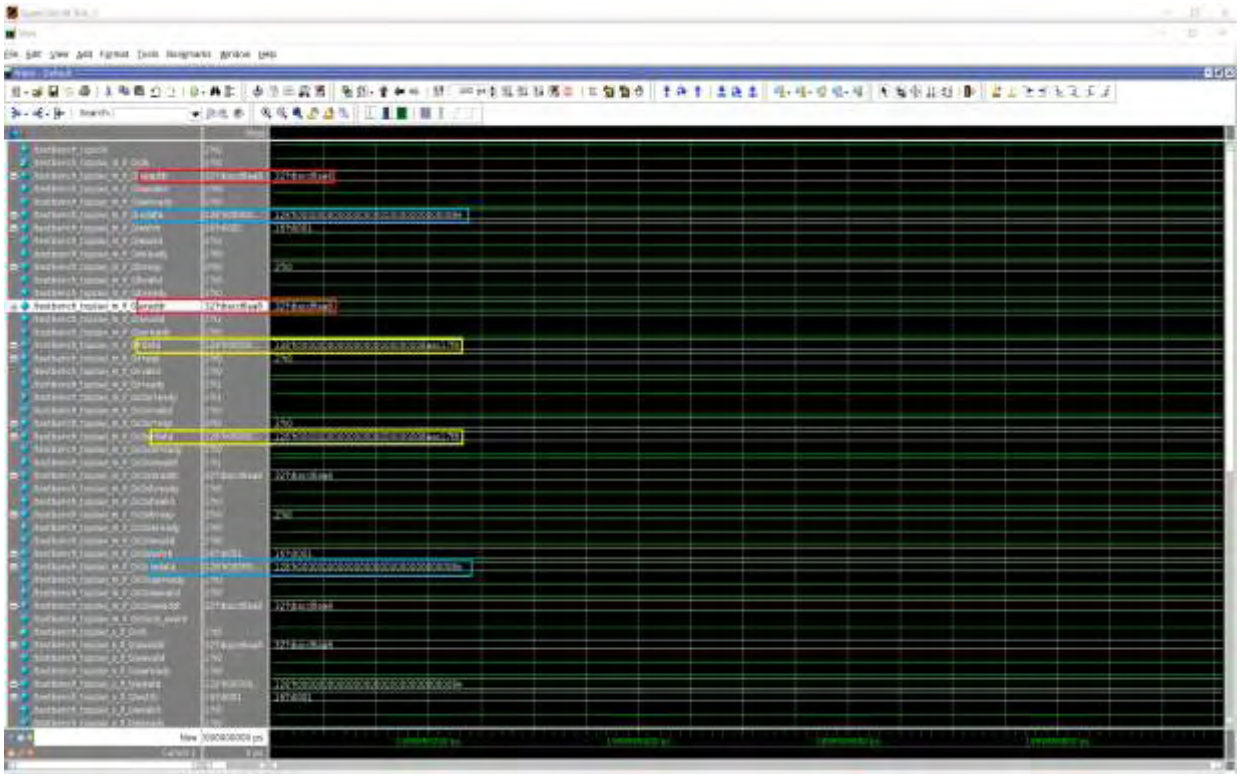

Figure 18: AXI4-Lite Read/Write Transactions

Values of Read and Write Data (DataOut, DataIn) are the same on both master and slave interfaces, thus we can conclude the transaction was successful.

AXI4-Lite simulation

For the simulation of the AXI4-Stream design the following test scenario was created:

- 1 AXI4- Stream Master interface
- 1 AXI4-Stream Slave interface
- 1 AXI4-Stream monitor
- Insertion of random delays at both Master and Slave
- Generation of Random packets
- Master creates a User buffer and sends data
- Slave reads data and outputs them to the Read User Buffer

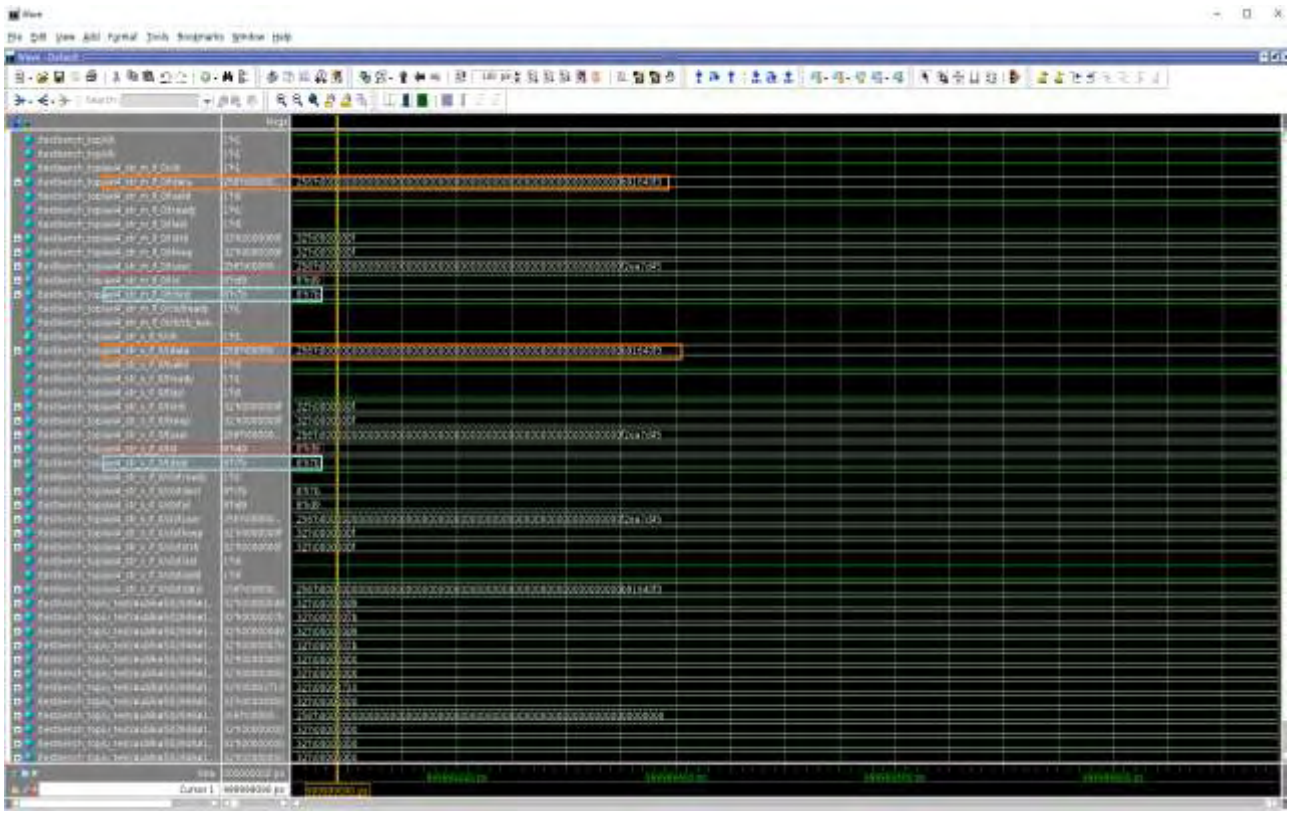

Figure 19: AXI4-Stream

The values of **TID, TDEST, TDATA** are the same on both master and slave interfaces, thus we can conclude the transaction was successful.

References

[1] AMBA® AXI™ and ACE™ Protocol Specification). Available at http://www.arm.com

[2] AXI4™ and AXI4-Lite™ protocol assertion descriptions. Available at http://www.arm.com

[3] Questa® SIM GUI Reference Manual Including Support for Questa SV/AFV. Software Version 10.4c